Fazliu Arber
Mace Pierrick

Compte rendu projet Algorithmique

I.Description Structures utilisées

Structures du tp1 utilisées:

```
struct _list {
   int startNode;         /* etat de depart de la transition */
   int targetNode;        /* cible de la transition */
   unsigned char letter;  /* etiquette de la transition */
   struct _list *next;    /* maillon suivant */
};
```

```
struct _trie {
   int maxNode;       /* Nombre maximal de noeuds du trie */
   int nextNode;      /* Indice du prochain noeud disponible */
   List *transition;  /* listes d'adjacence */
   size_t *finite;    /* etats terminaux */
};
```

Structures créées

Dans aho-corasick/ac.h :
struct aho_corasick est une structure contenant un trie et un tableau dynamique d'entiers.
C'est la structure qui sera manipulée par l'algorithme d'Aho-Corasick

```
typedef struct aho_corasick {
  Trie trie;
  int sup[];
} *AC;
```

Dans queue/queue.h :
struct queue permet de définir une file générique qui sera utilisée lors de la phase de prétraitement de l'algorithme d'Aho-Corasick.

```
typedef struct queue {
  size_t capacity;
  size_t head;
  size_t tail;
  void **queue;
} *Queue;
```

II.Description Fonctions utilisées

Module aho-corasick/ac.c

AC createAC(Trie trie) : Permet de créer un nouveau trie.
size_t getOccurencesAC(AC ac, FILE *file) : Renvoi le nombre d'occurrence
static void complete(AC ac) :
void disposeAC(AC* ac) : Permet de libérer l'espace mémoire alloué pour le trie passé en paramètre.

Module generators/gen_mots.c

void generate_text(char string[], int len, int alphabet_size) : Génère un mots de longueur len et dont la taille de l'alphabet est alphabet_size.
char generate_random_character(int alphabet_size) : Génère un caractère aléatoirement selon la taille de l'alphabet passé en paramètre.

Module generators/gen_texte.c

void generate_text(char string[], int len, int alphabet_size) : Génère un texte de longueur len et dont la taille de l'alphabet est alphabet_size.
char generate_random_character(int alphabet_size);

Module queue/queue.c

Queue newQueue(size_t capacity) : Permet d'allouer une nouvelle file en initialisant la capacité, le pointeur de tête, de queue.
bool isQueueEmpty(Queue q) : Vérifie si la file est vide.
void enQueue(Queue q, void *element) : Ajoute un élément à la file.
void *deQueue(Queue q) : Retire un élément le de la file.
void freeQueue(Queue q) : Libère les ressources allouées à la file.

Module main-ac.c

int insertWordsInTrie(Trie trie, FILE *file) : Permet d'insérer dans le trie les mots contenus dans le fichier file.
Int getFileSize(FILE *file) : Renvoie la taille du fichier file en bytes

III. Description des résultats

Dans le fichier results.odf on peut voir qu'en général le temps d'exécution en ms pour les matrices de transitions est nettement moins élevé que pour les tables de hachages.

Ci-dessous on peut voir que pour un texte de longueur 5 000 000, avec des tailles d'alphabet allant de 2 à 70 et des tailles de mots variant entre 5-15 jusqu'à 30-60 le temps d'exécution (en moyenne pour 100 itérations) de la matrice de transition va de 77 à 60 ms alors que pour les table de hachage de 242 à 324 ms.

	A	B	C	D	E
1	Text size	Alphabet size	Word length	Transition matrix	Hashtable
2	5 000 000	2	5-15	77	242
3	5 000 000	2	15-30	85	276
4	5 000 000	2	30-60	93	271
5	5 000 000	4	5-15	75	309
6	5 000 000	4	15-30	80	315
7	5 000 000	4	30-60	86	340
8	5 000 000	20	5-15	56	314
9	5 000 000	20	15-30	57	327
10	5 000 000	20	30-60	62	334
11	5 000 000	70	5-15	56	288
12	5 000 000	70	15-30	56	302
13	5 000 000	70	30-60	60	324

Visuellement on peut voir ci-dessous sur le diagramme de gauche qu'avec les matrices de transition le temps d'exécution d'Aho-Corasick selon la taille des mots et de l'alphabet ne de dépasse pas 100 ms.

On remarque aussi qu'avec cette dernière méthode, qu'à partir du 8e teste (5-15 de longueur de mot et une taille d'alphabet de 20) jusqu'au 13e test (30-60 longueur de mot et taille d'alphabet de 70) le temps d'exécution se stabilise proche de 50 ms. Alors qu'avec les tables de hachages le temps d'exécution monte jusqu'à 350 ms (7e test) et se stabilise proche de 300ms.

Le diagramme de droite nous montre aussi l'évolution du temps d'exécution de chacun des méthodes. Plus la taille de l'alphabet et des mots augmente, plus le temps d'exécution avec la table de hachage augmente : de 242 à 324 et se stabilise à 300 ms quand la taille de l'alphabet passe à 20.
Alors qu'avec les matrices de transition il reste stable puis décroît lorsque la taille de l'alphabet est de 4 et se restabilise à 50 ms lorsque la taille de l'alphabet passe à 20.